Introduction technique à XSLT

Code: xml-xslt

Originaux

url: **http://tecfa.unige.ch/guides/tie/html/xml-xslt/xml-xslt.html**

url: **http://tecfa.unige.ch/guides/tie/pdf/files/xml-xslt.pdf**

Auteurs et version

- Daniel K. Schneider - Vivian Synteta

- Version: 1.9 (modifé le 2/10/07 par DKS)

Prérequis

Module technique précédent: xml-dom (éléments du XML Framework)

Module technique précédent: xml-tech (arbres XML, DTDs)

Module technique précédent: xml-xpath (complément)

Modules suivants

Module technique suivant: xml-xslt2 (suite directe)

Module technique suivant: xml-xslfo

Objectifs

- Avoir une idée de XSLT 1.0
- Savoir écrire des simples feuilles de transformations XML vers (X)HTML

Notes:

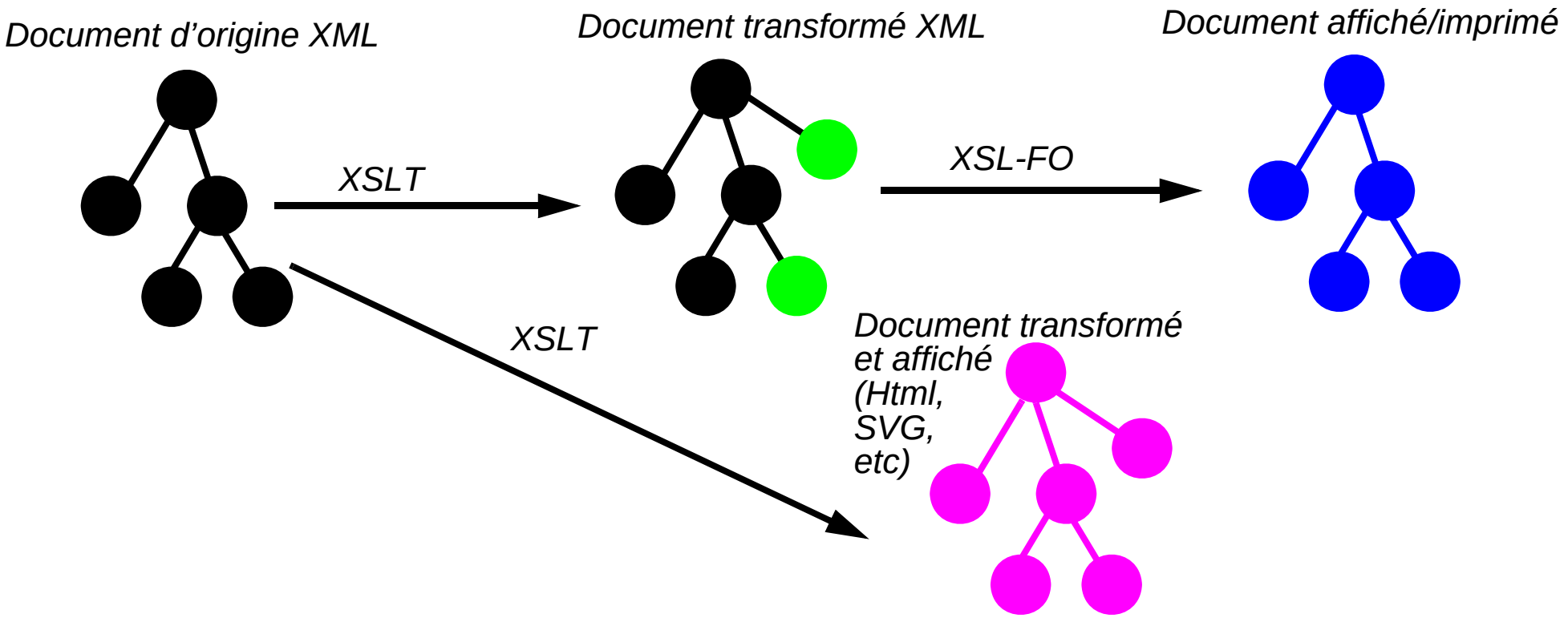
- Il reste pas mal d'améliorations à faire !!
- Limité à XSLT 1.0 (il existe déjà des processeurs pour XSLT 2.0)
- Il reste des lacunes ....

1. Table des matières détaillée

2. Introduction à XSLT

But de XSLT

- XSLT est un langage de transformations d'arbres (fichiers) XML
- XSLT est écrit en XML
- XSLT permet la génération d'autres contenus à partir d'un fichier XML, par exemple:
  - du HTML (bien formé)
  - du XML plus compliqué (tables de matière + règles de formattage XSL/FO)
  - des extraits en XML ou HTML ....
  - du SVG, X3D ou toutes sortes d'autres formats à partir de XML



Utilisation (mécanisme de base):

1. Définir des règles qui disent comment transformer un "noeud" (element, tag) et ses sous-éléments
2. Organiser l'application de ces règles

Les feuilles de style XSLT

- Une feuille de style XSLT est un document séparé qui contient des règles de transformation XSLT
- On peut associer une feuille de style XSL(T) à un (ou plusieurs) documents XML
- Marche soit en "batch" (dans un éditeur ou en ligne de commande), soit avec un traitement "server-side", soit avec la plupart des clients Web modernes (Mozilla, IE6, etc.)

Documentation

- La spécification de XSLT 1.0 est formalisée (W3C Recommendation 16/11/99)
  url: http://www.w3.org/TR/xslt
- A Tecfa: voir la page XML/XSL dans la toolbox

Compléments à XSLT

- Xpath (langage pour indiquer un chemin dans un arbre): **http://www.w3.org/TR/xpath**
- XSL/FO (mise en page): **http://www.w3.org/TR/xsl/**
- XQuery (langage d'interrogation, bases de données XML): , **http://www.w3.org/TR/xquery/**

2.1 Entêtes et utilisation des fichiers XSLT

A. Définition d'un fichier XSLT

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
....
</xsl:stylesheet>
```

- Normalement les fichiers XSLT ont l'extension *.xsl
- `xmlns:xsl="URL"` définit un "namespace" pour les balises XSL
- Donc dans nos exemples toutes les balises XSL commencent par `"xsl:"`
- Le fait que toutes les balises XSL commencent par `xsl:` empêche toute confusion

B. Association d'un fichier XSLT à un fichier XML

L'association peut se faire dans le fichier XML:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet href="project.xsl" type="text/xsl"?>
<votre_xml>
......
</votre_xml>
```

- Note: Il existe d'autres méthodes d'associations. Par exemple, dans un traitement "batch" on utilise une instruction comme `"saxon -o fichier.html fichier.xml fichier.xsl"` pour dire "utilise tel fichier ".xsl" pour tel fichier ".xml" pour produire tel fichier ".html".

2.2 Principe de fonctionnement de XSLT

- XSLT est un véritable langage de programmation

Utilisation simple de XSLT

- La transformation du document source (XML) se fait selon des règles (facteurs conditionnels)
- Une feuille de style XSL contient un jeu de règles qui déclarent comment traduire des éléments XML (selon leur contexte).
- On peut imaginer qu'on veuille traduire un commentaire défini par une balise XML `<commentaire>` en une construction html "`<dl>`" comme ci-dessous:

  L'expression XML suivante:

  ```
  <commentaire> xxxx </commentaire>
  ```

  pourrait donner:

  ```
  <DL>
       <DT>Commentaire  </DT>
       <DD> xxxx </DD>
   </DL>
  ```

- Le but d'une feuille de style XML serait de définir une transformation pour chaque balise XML

Une simple règle de traduction (appelée "template" en XSLT):

```
<xsl:template match="XML_tag_name">
  .... contents to produce (i.e. HTML)
  .... further instructions
</xsl:template>
```

Selector of XML elements to transform

Contenus à produire + autres instructions

Example 2-1: Traduction d'une balise "title" en "h1" centrée

Source XML à traduire:

```
<title>Hello friend</title>
```

La règle XSLT:

```
<xsl:template match="title">
   <h1 align="center">
      <xsl:apply-templates/>
   </h1>
</xsl:template>
```

Règle pour "title"

Code HTML produit, c.a.d la traduction !

Veut dire "continuer": chercher autre règle ou copier les contenus

2.3 Un simple exemple XSLT

Exemple 2-2: Sensibilisation XML + XSLT

url: **http://tecfa.unige.ch/guides/xml/examples/simple/hello-page.dtd (DTD)**

url: **http://tecfa.unige.ch/guides/xml/examples/simple/hello-page.xml.text (src XML)**

url: **http://tecfa.unige.ch/guides/xml/examples/simple/hello-page-html.xsl (style)**

url: **http://tecfa.unige.ch/guides/xml/examples/simple/hello-page.xml (XML)**

url: **http://tecfa.unige.ch/guides/xml/examples/simple/hello-page.html (résultat)**

Le fichier XML (sans feuille de style):

```
<?xml version="1.0"?>
<page>
 <title>Hello Cocoon friend</title>
 <content>Here is some content :)  </content>
 <comment>Written by DKS/Tecfa, adapted from S.M./the Cocoon samples </
comment>
</page>
```

Le "document" XHTML résultant que l'on désire obtenir:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN" "http://www.w3.org/TR/REC-html40/
strict.dtd">
<html><head><title>Hello Cocoon friend</title></head><body bgcolor="#ffffff">
 <h1 align="center">Hello Cocoon friend</h1>
 <p align="center"> Here is some content :) </p>
 <hr> Written by DKS/Tecfa, adapted from S.M./the Cocoon samples
  </body></html>
```

Le fichier XSL:

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="page">
.....
   <html> <head> <title> <xsl:value-of select="title"/> </title> </head>
    <body bgcolor="#ffffff">
     <xsl:apply-templates/>
    </body>
   </html>
  </xsl:template>

  <xsl:template match="title">
   <h1 align="center"> <xsl:apply-templates/> </h1>
  </xsl:template>

  <xsl:template match="content">
   <p align="center"> <xsl:apply-templates/> </p>
  </xsl:template>

  <xsl:template match="comment">
   <hr /> <xsl:apply-templates/>
  </xsl:template>
</xsl:stylesheet>
```

3. XSL de base

Opérations de base:

1. Définir pour chaque balise une règle qui traduit la balise et son contenu
2. Organiser l'application de ces règles, c.a.d. indiquer comment traiter le contenu.
3. Note: Rappelez-vous que XSL est du XML (donc il faut respecter les principes de validité et de bien formé (pas de balises croisés par exemple !)

Rappel définition d'une règle ("template") avec xsl:template

```
<xsl:template match="nom de la balise">
   contenus générés
   et autres instructions XSLT
</xsl:template>
```

Sélecteur d'éléments XML à transformer

Ici vont s'insérer toutes les balises générées et éventuelles instructions XSLT aboutissant à d'autres transformations

Exemple 3-1: Exemples xsl:template match="xxx"

Une règle applicable à toutes les balises "project":

```
<xsl:template match="project">
  ......
</xsl:template>
```

3.1 Anatomie d'une simple feuille de style

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://
www.w3.org/1999/XSL/Transform">
<xsl:template match="page">
   .......
 <html>
   <head> <title>
        <xsl:value-of select="title"/>
   </title> </head>
   <body bgcolor="#ffffff">
     <xsl:apply-templates/>
   </body>
   </html>
</xsl:template>

<xsl:template match="title">
   <h1 align="center"> <xsl:apply-templates/> </h1>
</xsl:template>
...........

</xsl:stylesheet>
```

Déclarations XSLT (début de la feuille de style)

Règle pour l'élément racine du document

Une autre règle

Fin de la feuille

3.2 Application de templates aux sous-éléments

A. <xsl:apply-templates />

- Ici on définit une simple règle pour la racine qui se déclenche (normalement) en premier):

```
<xsl:template match="/">
   <html> <body>
     <xsl:apply-templates/>
   </body> </html>
 </xsl:template>
```

- Un simple apply-templates (sans attributs) examine tous les noeuds enfants dans l'ordre. Si une règle qui correspond à un noeud est détectée, elle sera appliquée

```
<page>
<title>Hello Cocoon friend</title>
 <content>Here is some content :)  </content>
 <comment>Written by DKS/Tecfa, adapted from S.M./the Cocoon samples </
comment>
</page>
```

- Pour le XML ci-dessus, les 2 règles pour "title" et "content" se déclencheraient!

```
<xsl:template match="title">
   <h1 align="center"> <xsl:apply-templates/> </h1>
  </xsl:template>
<xsl:template match="content">
   <p align="center"> <xsl:apply-templates/> </p>
</xsl:template>
```

B. L'attribut "select" de apply-templates

- permet de spécifier un élément défini par un XPath (au lieu de tous les enfants),
- Autrement dit, on donne l'ordre explicite de chercher et d'appliquer toutes les règles à disposition pour un seul type d'élément identifé par un XPath
- Dans l'exemple ci-dessous la règle déclenchée pour un élément <page> lance seulement la règle qui s'applique au sous-élément <title>

```
<xsl:template match="page">
  <xsl:apply-templates select="title"/>
</xsl:template>
```

Cette règle pourrait s'appliquer au texte XML suivant:

```
<page>
 <title>Hello Cocoon friend</title>
 <content>
  Here is some content. Olé !
 </content>
 <comment>
      Written by DKS/Tecfa, adapted from S.M./the Cocoon samples
 </comment>
</page>
```

url: **http://tecfa.unige.ch/guides/xml/examples/simple/hello-page.xml**

3.3 Déroulement de l'exécution des règles

En simplifiant .....

- Le "moteur" XSLT cherche d'abord à exécuter la ***première*** règle qu'il trouve pour l'élément racine.
- Cette règle normalement fait appel à d'autres règles
  - soit implicitement : **`<xsl:apply-templates/>`**
  - soit en faisant appel à des règles précises: **`<xsl:apply-templates select="regle"/>`**
- Chacune des sous-règles qui peuvent s'appliquer sera exécutée dans l'ordre et ainsi de suite
- Le processeur ne trouve que les règles qui s'appliquent aux enfants du contexte actuel !!!

url: **http://tecfa.unige.ch/guides/xml/examples/simple/hello-page-wrong.xml**

```
<page>
 <title>Hello Cocoon friend</title>
 <content> <p> Here is some content. Olé ! </p> </content>
 <comment> Written by DKS/Tecfa, adapted from S.M./the Cocoon samples </comment>
</page>
```

La règle suivante ne marche pas, car **`comment`** n'est pas un enfant de **`content`**

```
<xsl:template match="content">
  <xsl:apply-templates select="comment"/>
</xsl:template>
```

La règle suivante marche:

```
<xsl:template match="content">
  <xsl:apply-templates select="p>
</xsl:template>
```

4. XPATH et extraction de valeurs

- Pour mieux comprendre le fonctionnement des "templates" et pour aborder l'instruction **`<xsl:value-of .... />`** (qui permet d'extraire des informations d'un document source), il faut avoir des notions du langage "XPath"...
- XPath permet d'identifier un ou plusieurs fragments d'un document XML (cela rappelle les sélecteurs de CSS)

Exemple d'un simple XPATH:

```
<xsl:template match="page">
  <xsl:apply-templates select="title"/>
</xsl:template>
```

"page" et "title sont des expressions XPath définissant un chemin de localisation "

```
<xsl:template match="content">
  <xsl:apply-templates select="page/comment"/>
</xsl:template>
```

"/page/comment" est une expression XPath un peu plus compliquée

XPath est nettement plus puissant que les sélecteurs CSS. Avec XPath vous pouvez par exemple dire "Identifiez le 4ème mot du 2ème paragraph qui suit un titre qui commence par le mot 'début'".

4.1 Récapitulatif de chemins simples de localisations XPATH

Voir aussi:

url: **http://tecfa.unige.ch/guides/tie/html/xml-path/xml-xpath.html**

Elément syntaxique	(Type de chemin)	Exemple d'un chemin	Exemple d'un match réussi par rapport au chemin indiqué à gauche
balise	nom d'élément	project	***<project> ..... </project>***
/	sépare enfants directs	project/title	<project>***<title> ... </title>***
		/	(correspond à l'élément racine)
//	descendant	project//title	<project><problem>***<title>....</title>***
		//title	<racine>...***<title>..</title>*** (n'mporte où)
*****	"wildcard"	*/title	<bla>***<title>..</title> et*** <bli>***<title>...</title>***
|	opérateur "ou"	title|head	<title>...</title> ***ou*** <head> ...</head>
		|/|@	(tous les éléments: les enfants, la racine et les attributs de la racine)
.	élément courant	.	
../	élément supérieur	../problem	<project>
@	nom d'attribut	@id	<xyz id="test">...</xyz>
		project/@id	***<project id="test" ...> ... </project>***
@attr='type'		list[@type='ol']	***<list type="ol"> ..... </list>***

4.2 Extraction d'une valeur

A. xsl:value-of

- Sélectionne le résultat d'un XPath et le copie vers le document "sortie"
- Autrement dit: on extrait le contenu d'sous-élément, la valeur d'un attribut, etc.

Exemple:

- La règle suivante se déclenche dès qu'une balise <projet> est trouvée
- Elle insère dans le document de sortie le contenu de l'élément <title> qui se trouve à l'intérieur d'un sous-élément <problem>

```
<xsl:template match="project">
    <P>
      <xsl:value-of select="problem/title"/>
    </P>
  </xsl:template>
```

Syntaxe spéciale pour insérer la valeur d'un objet dans le string d'un attribut à générer:

Syntaxe: {....}

```
<xsl:template match="contact-info">
....
  <a href="mailto:{@email}"><xsl:value-of select="@email"/></a>
...
```

{@email} insère la valeur de l'attribut email de l'élément courrant, par exemple:

```
<contact-info email="test@test">
```

B. xsl:copy

- Sert à copier "tel quel" un élément source vers le document produit
- Copie les balises et le contenu !

Scénarios d'usage

- Utile pour reproduire l'original (ici un tag ...)

```
<xsl:template match="p">
 <xsl:copy> <xsl:apply-templates/> </xsl:copy>
</xsl:template>
```

- Utile pour récupérer tout ce qui n'a pas été défini, mais attention si le tag ne correspond pas à un tag HTML il faut regarder le source HTML produit et agir ....

```
<xsl:template match="*">
   <xsl:copy>Garbage: <i> <xsl:apply-templates/> </i> </xsl:copy>
</xsl:template>
```

- Le fragment suivant copie tous les éléments non traités par les autres templates:
  - utile si vous utilisez des balises XHTML que vous ne désirez pas traiter ou encore si XSLT sert juste à "enrichir" votre code XML (genre table de matières)

```
<xsl:template match="*|@*">
  <xsl:copy>
   <xsl:apply-templates select="@*"/>
   <xsl:apply-templates select="node()"/>
  </xsl:copy>
 </xsl:template>
```

5. Exemples

5.1 Utilisation de apply-templates et XPath

- Avant d'utiliser des constructions avancées comme "if" ou "for-each", réfléchissez bien si c'est vraiment nécessaire. Dans la plupart des cas il suffit de définir des règles avec "xsl:apply-templates" et le reste "s'organise" tout seul.

Exemple 5-1: Simple XML vers HTML avec XSLT

url: **http://tecfa.unige.ch/guides/xml/examples/xsl-simple/ (fouiller le répertoire !)**

A. Un texte en XML

```
<arbre>
 <para>Simples Templates et XPath</para>
 <aunt>
  <name>Auntie</name>
  <child>Je suis un enfant de aunt</child>
 </aunt>
 <uncle>
  <name>Uncle Ben</name>
  <child>Je suis le premier enfant de uncle</child>
  <child>Je suis le 2eme enfant de uncle</child>
  <child>Je suis le 3eme enfant de uncle</child>
 </uncle>
</arbre>
```

B. La feuille de style XSLT

- Ici on veut produire un simple HTML à partir du XML, avec les consignes suivantes:
  - On aimerait que les enfants de <aunt> et <uncle> soient affichés différemment
  - Le prermier enfant de <uncle> doit être affiché spécialement aussi
- Si voulez savoir comment ce code s'exécute, consultez le fichier *-trace.xml

```
<xsl:template match="arbre">
 <html><title>XSL Example</title><body>
   <xsl:apply-templates />
  </body> </html>
</xsl:template>

<xsl:template match="uncle|aunt">
  <hr /> <xsl:apply-templates />
</xsl:template>

<xsl:template match="name">
 <xsl:apply-templates /> :
</xsl:template>

<xsl:template match="uncle/child[position()=1]">
 <p> <strong><xsl:apply-templates /></strong> </p>
</xsl:template>
<xsl:template match="uncle/child[position()>1]">
 <p> <xsl:apply-templates /></p>
</xsl:template>
<xsl:template match="aunt/child">
 <p style="color:blue"><xsl:apply-templates /></p>
</xsl:template>
```

C. Le résultat en HTML

```
<!DOCTYPE html
  PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
   <title>XSL Example</title>
   <body>
      Simples Templates et XPath
      <hr>
      Auntie :
      <p style="color:blue">Je suis un enfant de aunt</p>
      <hr>
      Uncle Ben :
      <p><strong>Je suis le premier enfant de uncle</strong></p>
      <p>Je suis le 2eme enfant de uncle</p>
      <p>Je suis le 3eme enfant de uncle</p>
   </body>
</html>
```

5.2 Gestion de liens

Le formalisme XML en soi ne comprend pas les liens !!

- Il existe un langage XLink que vous pouvez utiliser (et qui fait partie des standards SVG)
- Sinon, vous inventez un élément ou encore un attribut pour ce type d'information et vous traduisez en <a href ...>

Exemple 5-2: Traduction vers ...

url: **http://tecfa.unige.ch/guides/xml/examples/xsl-links-to-url/links.xml**

url: **http://tecfa.unige.ch/guides/xml/examples/xsl-links-to-url/links.xsl**

c.f. le code source de ces fichiers !

A. Extraits du fichier XML

- On peut inventer une balise comme "url" pour l'URL et une autre balise comme "name" pour indiquer le nom du lien.

```
<address>
  <name>TECFA</name>
  <url>http://tecfa.unige.ch</url>
 </address>

 <address2>
  <name>TECFA</name>
  <url>http://tecfa.unige.ch</url>
 </address2>
```

- Alternativement, on aurait pu mettre des informations dans un attribut

```
<address4 url="http://tecfa.unige.ch">TECFA</address4>
<address5 url="http://tecfa.unige.ch">TECFA</address5>
```

B. Extraits du fichier XSLT

Pour address et address2 on crée des règles

```
<xsl:template match="address">
  <a href="{url}"> <xsl:value-of select="name"/> </a>
</xsl:template>

<xsl:template match="address2">
    <xsl:apply-templates select="url"/>
</xsl:template>

<xsl:template match="url">
 <a href="{.}"> <xsl:value-of select="../name"/> </a>
</xsl:template>
```

address3 et address2 sont traitées directement depuis l'élément mère

```
<a href="{address3/url}"><xsl:value-of select="address3/name"/></a>
<a href="{address4/@url}"><xsl:value-of select="address4"/></a>

<xsl:template match="address5">
  <a href="{@url}"> <xsl:value-of select="."/> </a>
</xsl:template>
```

5.3 Images

- Il y n'a aucune magie spéciale pour gérer les images !
- Simplement:
  - Examinez votre XML
  - Trouvez un moyen pour traduire en HTML (ou autre chose)

Example 5-3: Insertion d'images

Fichier XML

```
<?xml version="1.0"?>
<?xml-stylesheet href="images.xsl" type="text/xsl"?>
<page>
 <title>Hello Here are my images</title>
 <list>
   <image>dolores_001.jpg</image>
   <image>dolores_002.jpg</image>
   <image>dolores_002.jpg</image>
   <image2>scrolls.jpg </image2>
   <image2>scrolls.jpg </image2>
   <image3 source="dolores_002.jpg">Recipe image</image3>
 </list>
 <comment>Written by DKS.</comment>
</page>
```

XSLT stylesheet

- Une règle pour la balise "list"

```
<xsl:template match="list">
   Apply templates for "image" elements:
   <xsl:apply-templates select="image"/>

   This will only insert the first "image2" element contents it finds:
   <p> <img src="{image2}"/> </p>

   And another template for a tag image3 element (with an attribute)
   <xsl:apply-templates select="image3"/>
</xsl:template>
```

- Une règle pour la balise "image"

```
<xsl:template match="image">
  <p> <img src="{.}"/> </p>
</xsl:template>
```

- Une règle pour la balise "image3"

```
<xsl:template match="image3">
  <p> <img src="{@source}"/><xsl:value-of select="."/> </p>
</xsl:template>
```

5.4 Fabrication de références (liens)

Exemple 5-4: Table de matières pour éléments qui ont un identificateur

url: **http://tecfa.unige.ch/guides/xml/examples/recit/**

- On utilise ici un attribut "mode" dans la définition des templates et apply-templates
- Cela nous permet d'écrire plusieurs règles pour un même noeud
- Ici par exemple, on utilise mode="toc" pour fabriquer une table des matières

Fragment XSLT (fait d'abord la table des matières, ensuite le reste)

```
<xsl:template match="/">
  <html>
   <body bgcolor="#FFFFFF">
     <h1><xsl:value-of select="/RECIT/Titre"/></h1>
     <p> Highlights de l'histoire:
         <xsl:apply-templates select="//EPISODE" mode="toc"/> </p>
     <xsl:apply-templates/>
   </body>
 </html>
</xsl:template>
```

Fragments XSLT pour gérér la balise "EPISODE": 2 règles dont une avec un mode

```
<xsl:template match="EPISODE" mode="toc">
  <a href="#{@id}"><xsl:value-of select="SousBut"/></a> -
</xsl:template>
<xsl:template match="/RECIT/FIL/EPISODE">
  <a name="{@id}"> <hr /> </a>
  <xsl:apply-templates/>
</xsl:template>
```

Exemple 5-5: Tables de matières pour éléments sans ID

- plus difficile car il faut "fabriquer" des attributs "name" et "href" pour le HTML
  - la solution adopté ici est moche (on aurait pu compter les éléments)
- il s'agit des jours d'un atelier webmaster qu'on peut consulter ici:

  url: **http://tecfa.unige.ch/tecfa/teaching/formcont/webmaster2003/programme/**
- Pour comprendre cet exemple il faut fouiller dans les fichiers suivants:
  - programme.xml - contient tout le "programme" de l'Atelier
  - programme.xsl - produit le résultat (voir le fichier programme.html)
  - matos.xsl, resume.xsl, animateurs.xsl créent des extraits variés du programme.

Fragments XSLT

```
<!-- Code qui fabrique la table des matières (jours) en HTML -->
Programme: <xsl:apply-templates select="//day" mode="toc" />

<xsl:template match="day" mode="toc">
  <a href="#{@name}{@dayno}{@month}"><xsl:value-of select="@name"/></a> -
</xsl:template>
<!-- Code pour insérer des attributs "name" dans le HTML -->
<xsl:template match="day">
   <a name="{@name}{@dayno}{@month}"><xsl:value-of select="@name"/></a> -
   <xsl:value-of select="@dayno"/>/<xsl:value-of select="@month"/>/
   <xsl:value-of select="@year"/>
</xsl:template>
```

Fragment XML (élément à extraire)

- Faire une table des matières avec les jours de l'Atelier (avec liens)

```
<day name="lundi" year="2003" month="6" dayno="16">
```

Résultat HTML

- Au début du fichier on a un menu qui affiche les jours (liens vers le bas)

```
Programme: <a href="#lundi166">lundi</a> - <a href="#mardi176">mardi</a> - ....
.......
```

- Dans le fichier on insère les attributs "name"

```
<a name="lundi166">lundi</a> - 16/6/2003
```

Note:

- Dans ce répertoire il y a aussi un fichier programme-fo.xsl qui génère du code xsl-fo utilisé pour générer la version PDF du programme.

Exemple 5-6: Tables de matières pour éléments sans ID

- Table des matières pour une "page travaux STAF" (portfolio étudiant)
- On liste les travaux

  url: **http://tecfa.unige.ch/guides/xml/examples/xsl-toc/**

6. Déclarations et style

6.1 Déclaration de la sortie

- Veillez à ce que le fichier produit corresponde aux normes, c.à.d qu'il contienne les déclarations nécessaires pour chaque type de document.

xsl:output

permet de définir le type de sortie qui sera produit et de générer des entêtes. Voici la syntaxe (simplifiée) pour XSL V 1.0 (1999)

```
Syntaxe: <xsl:output
     method = "xml" | "html" | "text"
     version = nmtoken
     encoding = string
     omit-xml-declaration = "yes" | "no"
     standalone = "yes" | "no"
     doctype-public = string
     doctype-system = string
     indent = "yes" | "no"
     media-type = string />
```

- A mettre au début du fichier (après xsl:stylesheet)
- Ci-dessous qqs. exemples

Exemple 6-1: Output en HTML 4.01 transitionnel

```
<xsl:output method="html"
    encoding="ISO-8859-1"
    doctype-public="-//W3C//DTD HTML 4.01 Transitional//EN"/>
```

Exemple 6-2: Output en XHTML "façon light"

```
<xsl:output
   method="html"
    doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"
   doctype-public="-//W3C//DTD XHTML 1.0 Transitional//EN"
   indent="yes"
   encoding="iso-8859-1"
/>
```

Exemple 6-3: Output en XHTML "pur" (page XML) et transitionnel

```
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns="http://www.w3.org/1999/xhtml">
<xsl:output
   method="xml"
   doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"
   doctype-public="-//W3C//DTD XHTML 1.0 Transitional//EN"
   indent="yes"
   encoding="iso-8859-1"
/>
```

Exemple 6-4: Output en XHTML "pur" et strict

```
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns="http://www.w3.org/1999/xhtml">
<xsl:output
   method="xml"
   doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"
   doctype-public="-//W3C//DTD XHTML 1.0 Strict//EN"
   indent="yes"
   encoding="iso-8859-1"
/>
```

Exemple 6-5: Output en SVG

```
<xsl:output
 method="xml"
 indent="yes"
 standalone="no"
 doctype-public="-//W3C//DTD SVG 1.0//EN"
 doctype-system="http://www.w3.org/TR/2001/PR-SVG-20010719/DTD/svg10.dtd"
 media-type="image/svg"
/>
```

Exemple 6-6: Output en VRML

```
<xsl:output method="text"
 encoding="UTF-8"
 media-type="model/vrml"/>
....
<xsl:template match="/">#VRML V2.0 utf8
......
```

6.2 CSS pour le résultat de la transformation

- Lorsque vous produisez du HTML ou du XHTML, évitez de produire du HTML "vieille école" (balises "font", etc.)
- Il est simple d'associer une feuille de style CSS ! Vous "faites" comme à la main. Donc il faut insérer la balise "link" à l'endroit ou le <head> est généré.

```
<xsl:template match="racine">
 <html>
  <head>
   <link href="programme.css" type="text/css" rel="stylesheet"/>
   <title>
     bla bla
   </title>
  </head>
```

6.3 Générer plusieurs fichiers HTML à partir d'un seul XML

- Marche uniquement lorsqu'on traite le fichier XML en "batch"

Exemple 6-7: Programme de l'Atelier WebMaster 2004

url: **http://tecfa.unige.ch/tecfa/teaching/formcont/webmaster2004/programme/**

url: **programme.xsl (montre comment faire)**

```
<!-- au début du fichier -->
<xsl:output name="daypage" method="html" encoding="ISO-8859-1"
     doctype-public="-//W3C//DTD HTML 4.01 Transitional//EN"/>
.....
<!-- une règle qui génère une page -->
 <xsl:template match="software">
  <xsl:result-document href="software.html" format="daypage">
  <html> <head>
    <title> <xsl:value-of select="/course/course-title"/> </title> </head>
    <body bgcolor="white">
     [<a name="top" href="../welcome.html">Home</a>] -&gt; [<a name="top"
href="programme.html">Programme</a>] -&gt; [Logiciels]
     <ul>
      ......
      <xsl:apply-templates select="soft"/>
     </ul>
    </body>
   </html>
  </xsl:result-document>
 </xsl:template>
```

Quelques astuces:

- Si vous devez générer plusieurs pages pour un élément du même nom, il faut générer des noms de fichiers uniques:

```
<xsl:template match="day">
  <xsl:result-document href="{@name}{@dayno}{@month}.html" format="daypage">
```

  - dans l'exemple présent, on désire avoir une page / jour du programme. Chaque jour se distingue par une date spéciale (attributs dayno et month)

- Pour faire une table de matière (que vous pouvez inclure partout):
  - Voici la règle qui génère la table:

```
<xsl:template match="day" mode="toc">
 <a href="{@name}{@dayno}{@month}.html"><xsl:value-of select="@name"/></a> -
</xsl:template>
```

  - Voici comment l'inclure:

```
    <b>Programme</b>: <a href="programme.html"> Top</a> - <xsl:apply-
templates select="//day" mode="toc"/>
```

7. XSLT en "batch" et debugage

Avec un fichier de commande

- Il existe plusieurs processeurs XSLT populaires qu'on peut utiliser pour produire un fichier de sortie à partir d'un xml + xslt. On conseille Saxon (ci-dessous)
- Il faut passer par un processeur XSLT pour "debuger" une feuille de style XSLT client-side
  - "View source" dans un navigateur ne montre pas le HTML !
  - Donc maîtriser une "procédure manuelle" est intéressant !!
  - Alternativement, il existe des outils de debugage pour certains éditeurs
- Certains outils nécessitent l'installation d'un environnement Java.
  - il faut installer un runtime ou dévelopment kit Java pour utiliser les processeurs Xalan ou Saxon.
  - Un engin Runtime ("JRE") suffit:
  - Versions JRE 5 ou mieux !

  url: **http://java.sun.com/products/**

Avec un éditeur XML

- La plupart des éditeurs XML ont un processeur XSLT intégré (les versions commerciales ont des fonctions de debugage).

8. Server-side avec PHP

Exemple 8-1: XSLT avec PHP 5

- Le support XSLT est standard dans PHP 5
- L'exemple ci-dessous applique une feuille de style travaux.xsl au fichier travaux.xml

url: **http://tecfa.unige.ch/guides/php/examples/xslt/php-xslt.php**

url: **http://tecfa.unige.ch/guides/php/examples/xslt/php-xslt.phps**

url: **http://tecfa.unige.ch/guides/php/examples/xslt/php-xslt.text**

```
$xml_file = 'travaux.xml';
$xsl_file = 'travaux.xsl';
// load the xml file (and test first if it exists)
$dom_object = new DomDocument();
if (!file_exists($xml_file)) exit('Failed to open $xml_file');
$dom_object->load($xml_file);
// create dom object for the XSL stylesheet and configure the transformer
$xsl_obj = new DomDocument();
if (!file_exists($xsl_file)) exit('Failed to open $xsl_file');
$xsl_obj->load($xsl_file);
$proc = new XSLTProcessor;
$proc->importStyleSheet($xsl_obj); // attach the xsl rules
$html_fragment = $proc->transformToXML($dom_object);
print ($html_fragment);
```

- Donc pour utiliser ce "service", il suffit de copier le fichier php-xslt.php et changer les 2 noms de fichiers (travaux.xml et travaux.xsl) au début
  (enfin depuis le web il faut prendre php-xslt.text et le renommer en xxx.php)

9. Client-side XML+XSLT avec Firefox 1.0+ ou IE6+

- XSLT fonctionne sans problème avec les navigateurs IE, Mozilla/Firefox et Opera.

Exemple 9-1: Un simple exemple

url: **http://tecfa.unige.ch/guides/xml/examples/simple/hello-page.dtd (DTD)**

url: **http://tecfa.unige.ch/guides/xml/examples/simple/hello-page.xml.text (src XML)**

url: **http://tecfa.unige.ch/guides/xml/examples/simple/hello-page-html.xsl (style)**

url: **http://tecfa.unige.ch/guides/xml/examples/simple/hello-page.xml (XML)**

url: **http://tecfa.unige.ch/guides/xml/examples/simple/hello-page.html (résultat)**

- Il faut ajouter au moins la déclaration suivante (méthode de sérialisation)

```
<xsl:output method="html"/>
```

- Voir les fichiers pour les détails. Si aucun contenu n'est affiché ou s'il est mal affiché, faites "Menu->View->Source"
- Notes:
  - http://www.mozilla.org/releases/ (Mozilla downloads)
  - Evitez de travailler avec IE 5.5. La version "normale" n'est pas conforme au standard XSLT. Installez soit IE 6.x ou 7.x soit MSXML3+,pour IE 5.5.
  - Il faut s'assurer que le serveur Web indique le bon mime type pour xml et xsl (text/xml). Ceci est fait à Tecfa, mais pas forcément chez votre fournisseur....

10.Executive summary

1. Create a XSLT stylesheet file: xxx.xsl
2. Copy/paste the XSLT header and root element below (decide encoding as you like)

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">


</xsl:stylesheet>
```

3. Write a rule that deals with your XML root element
   - This rule must produce the root, head and body of the HTML (copy/paste this too)

```
<xsl:template match="page">
  <html>
  <head> <title> <xsl:value-of select="title"/> </title> </head>
  <body bgcolor="#ffffff">
     <xsl:apply-templates/>
  </body>
  </html>
</xsl:template>
```

4. Write rules for **each** (!!) of your XML elements,
   - for each insert some HTML, sometimes some text, or sometimes nothing
   - make sure to place a <xsl:apply-templates> inside each rule (usually between some HTML) ... unless you wish to censor contents.
5. Associate this stylesheet with your XML file using:
   - **<?xml-stylesheet href="xxx.xsl" type="text/xsl"?>**